A low-memory alternative for time-dependent Dijkstra

Simon Van den Eynde* Jeroen Verbrugghe Pieter Audenaert* Ben Derudder** Didier Colle* Mario Pickavet*

Abstract— **Time-dependent routing algorithms are a fundamental tool for calculating the fastest routes in road networks since the travel time of each road varies by departure time, due to congestion. While the time-dependent variant of Dijkstra's algorithm (TD-Dijkstra) can solve the routing problem optimally, it requires a large amount of memory. This paper presents a new memory-efficient time-dependent routing heuristic: the Time-Location Penalty Model (TLPM). Compared to time-independent Dijkstra, TLPM significantly increases accuracy in time-dependent routing problems, while keeping runtime and memory usage low.**

I. Introduction

Congestion is a problem, not only for drivers but also for routing algorithms. Therefore, we propose a new time-dependent routing heuristic: TLPM. This heuristic can calculate approximate fastest routes for any time of the day.

Since time-dependent routing has several real-life applications, much research has been done to design fast and accurate time-dependent routing algorithms. Unfortunately, while high accuracies can be reached, many of the previously presented algorithms require expensive computers, expensive programmers or both. TLPM is simple, fast and memory-efficient. Low memory requirements are especially important with the rise of online devices, such as smartwatches and cycle computers. An algorithm with lower memory requirements can work more easily without internet connection and on cheaper units. Recently, a survey on route planning in transportation networks was conducted [1]. One section of this survey details the recent advances in routing algorithms for time-dependent graphs. This section outlines a rich variety of algorithms, each balancing between memory requirements, preprocessing time and query time.

One of the most frequently used algorithms in this area is time-dependent contraction hierarchies (CH) [2], [3]. As a preprocessing step, CH sorts vertices in order of importance then contracts the vertices, least important first. To run a query, CH runs a bidirectional Dijkstra on the contracted graph. The query times are a factor 1000 faster than Dijkstra. However, CH requires a long preprocessing time and a lot of extra memory.

A more recently developed routing algorithm is Customizable Route Planning (CRP) [4]. This algorithm follows separator-based strategies. First, it layers the network from local to high-level, for example, the highest level usually contains highways. Then, these layers are connected. During the preprocessing of CRP, CH is used, while for querying CRP relies on bidirectional Dijkstra. The full CRP algorithm contains many more optimizations, which make the algorithm difficult to implement as efficiently as the author's implementation. However, their techniques are valuable. By separating the preprocessing step in metric-independent preprocessing and metric customization, the properties of the metric can be changed quickly. This increases the versatility of CRP: it is for example possible to integrate turn costs at negligible computational overhead. Furthermore, while their query times do not match those of their competitors, they are sufficiently low for real-time networks. But, since at its core CRP uses CH, CH's time and memory complexities are transferred.

Both CH and CRP are exact algorithms, but heuristic variants with good bounds are developed often. A recently proposed heuristic is Time-Dependent Simple routing (TD-S) [5]. At preprocessing, several time-windows are defined. Then, for each of these time-windows, a static graph is generated. At each query, TD-S first calculates the shortest path in each of the static graphs, subsequently creating a subgraph of the time-dependent graph with all the edges from these paths. Lastly, TD-S runs a time-dependent heuristic on the subgraph. To ensure that the search in the static graphs runs fast, they are preprocessed with the time-independent variant of CH.

The algorithm we propose, TLPM, is a heuristic - it will not produce exact results. Moreover, TLPM is a prototype: while its structure is sound, integrations with advanced speed-up techniques - such as CH - could drastically decrease query time. Currently, the implementation has the same runtime as Dijkstra. However, TLPM already exhibits several compelling properties. TLPM requires minimal preprocessing time, hardly requires extra memory (about one number per edge) and is simple to implement. Thus, it is suited for offline, low-memory devices. As such, TLPM can be seen as an alternative to time-dependent Dijkstra if the memory requirements do not allow saving all the time-data tables.

The next section begins by introducing the formal notation necessary to describe time-dependent road networks. This paper will then go on explaining time and location penalties to show how these can be combined to construct TLPM. The fourth section discusses the methodology used for this study. In particular, it explains how differential evolution finds good parameters for TLPM. The last two sections describe the results and summarize the conclusions.

II. Preliminaries

In this section, we lay out the necessary definitions and explanations to describe and understand time-dependent rout-

* Ghent University - imec, IDLab
** Ghent University

ing. Next, the data collection and processing is reported. Finally, it is demonstrated that, with adequate data preparation, TD-Dijkstra returns routes with minimal travel times.

A. Definitions

If V is the set of road network intersections (nodes) and E the set of road network edges, the road network is defined by $G(V, E)$. We write $\mathbb{R}^n_{\geq 0}$ for $\{x \in \mathbb{R}_{\geq 0} : x < n\}$. To model the time-dependency, we construct the function $\tau : E, \mathbb{R}^{1440}_{\geq 0} \to \mathbb{R}_{\geq 0}$ that maps an edge e and a departure time t (in minutes) to the time it takes to travel across e, departing at t.

Let $\tau_{min}(e)$, $e \in E$ be the minimal time it takes to travel along this edge e, expressed in seconds. This minimal time is generally equivalent to the travel time when driving alone at the speed limit with all the traffic lights on green. Since our time-data is a table with travel times for each edge and minute, τ will be a piecewise linear function with at most 1440 pieces.

The delay at edge e and departure time t is $\tau(e,t) - \tau_{min}(e)$, so the delay is the difference between the actual and minimal travel time. The loss at e and t is defined as the delay divided by $\tau_{min}(e)$. Thus, a loss of 0 means we have no delay, while a loss of 1 means it takes twice as much time to cross e compared to the minimal travel time. Since the delay depends on the minimal travel time of the edge, we cannot use it to compare edges. By contrast, the loss can be used therefor.

B. Data preprocessing

Our dataset contains a routing network that is located in East-Flanders, a Belgian province with around 1.5 million residents and an area of 2991 km^2 (Be-Mobile 2014). Each edge has a minimal travel time and a list of actual travel times, consisting of 1440 values: one for each minute of the day - a Tuesday in 2014. A section of Ghent, the capital and largest city of East-Flanders, is shown in fig. 1. While Ghent has a population of 260.000, the section we consider has an area of 90 km^2 and an estimated population of 160.000. In fig. 1 we see all the nodes of our dataset. Although each street contains many nodes - both at intersections and straight sections - not all streets of Ghent are represented in the dataset. In particular, most small streets are omitted, due to a lack of reliable data. Thus the dataset will seldom display cut-through driving.

After processing this data, making sure that every node is (part of) a real-life intersection, combining the data on the edges and efficiently storing all the time-data we modeled East-Flanders in a graph with 3825 nodes and 8570 time-dependent edges.

We train our model on the section of Ghent, which contains a great variety of location penalties. The Ghent section consists of 415 nodes and 833 edges.

C. FIFO-property

To calculate the fastest route between two nodes in a weighted graph, Dijkstra's algorithm (Dijkstra) is commonly

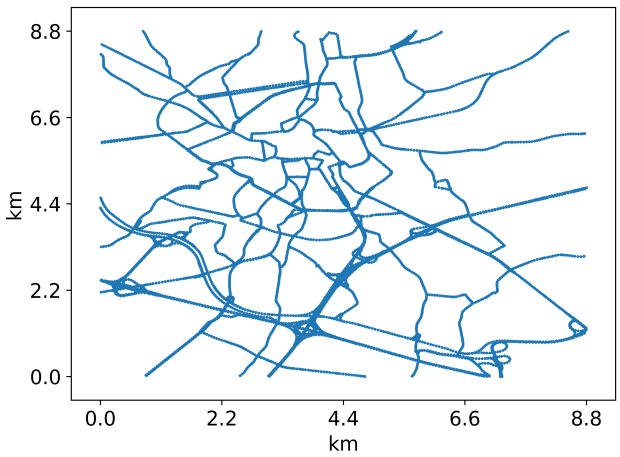


Fig. 1: The nodes of the Ghent section before preprocessing

used [6]. Dijkstra associates a weight (travel time) with each edge through a cost function $f_1 : E \to \mathbb{R}_{\geq 0}$. When the weights change in function of the departure time of the edge $\tau : E \times \mathbb{R}^{\leq 1440}_{\geq 0} \to \mathbb{R}$, one can switch to the time-dependent variant of Dijkstra (TD-Dijkstra) [7]. Compared to the time-independent algorithm, TD-Dijkstra can be implemented with little overhead. Furthermore, it guarantees to return the path with minimal travel time if each weight-function in our network fulfills the FIFO-property, that is $\forall t, t' \in \mathbb{R}^{\leq 1440}_{\geq 0}, \forall e \in E$:

$$t \leq t' \implies t + \tau(e,t) \leq t' + \tau(e,t')$$

in other words, if two cars A and B drive on the same edge and A leaves first, then B cannot arrive before A.

The high variation of the loss of an edge during the day makes the FIFO-requirement non-trivial. Thus, the travel times of the network are adapted to have the FIFO-property by limiting the downfalls: for every edge e and minute t, we redefine $\tau(e, t+1)$ sequentially:

$$\tau(e, t+1) \coloneqq \max\left[\tau(e, t+1), \tau(e,t) - 1\,\text{minute}\right]$$

Since our network now has the FIFO-property, TD-Dijkstra is guaranteed to return the fastest routes. Thus, we will consider the travel times for these routes as the goal of our model.

III. The TLPM algorithm

This section presents the Time-Location Penalty Model, a memory-efficient time-dependent routing algorithm. In particular, TLPM provides an improvement in accuracy for a minimal amount of extra memory. The basis of our model is the TD-Dijkstra algorithm, but with an adapted weight function based on the following observation: during rush hours almost all roads are congested, while at night almost none are.

This indicates that neither the road location nor the time of the day are good predictors for congestion by themselves, but that we require a combination of both. Thus, for each road, we calculate its congestion-susceptibility (location penalty)

TABLE I: Model descriptions: L and T stand for the Location and Time penalty, while the small letters are parameters.

Number	Description
0	0
1	$a \cdot L \cdot T$
2	$a + b \cdot L \cdot T$
3	$a + b \cdot L + c \cdot T + d \cdot L \cdot T$
4	$a + b \cdot L + c \cdot T + d \cdot L \cdot T + e \cdot L^2 + f \cdot T^2$
5	$a + b \cdot (c + L^d) \cdot (e + T^f)$

and for each moment in time the congestion-likelihood of the road network (time penalty). Our prediction of the loss at a road and a departure time then consists of a combination of the two associated penalties.

A. *Model formulas*

The location penalty of an edge e is $L(e)$: The average loss on e during the day.

$$\begin{aligned} L(e) &= \frac{1}{1440} \sum_t \left(\frac{\tau(e,t) - \tau_{min}(e)}{\tau_{min}(e)} \right) \\ &= \frac{1}{1440} \sum_t \left(\frac{\tau(e,t)}{\tau_{min}(e)} \right) - 1 \end{aligned}$$

The time penalty at time t is $T(t)$, the average loss on the entire network at time t.

$$\begin{aligned} T(t) &= \frac{1}{|E|} \sum_e \left(\frac{\tau(e,t) - \tau_{min}(e)}{\tau_{min}(e)} \right) \\ &= \frac{1}{|E|} \sum_e \left(\frac{\tau(e,t)}{\tau_{min}(e)} \right) - 1 \end{aligned}$$

For every edge, we save its location penalty. For each minute, we calculate the minute's time penalty, so we have - independent of the network size - 1440 time-penalties. Both formulas can be computed with a single linear pass through the data. The location penalty distribution in fig. 2b shows that many edges have a low penalty, but that some edges have a very high penalty - up to 3. The time penalties in figs. 3a and 3b on the other hand switch between 0 at night to 0.3 at day-time and reach a maximum of 0.6 during rush hours.

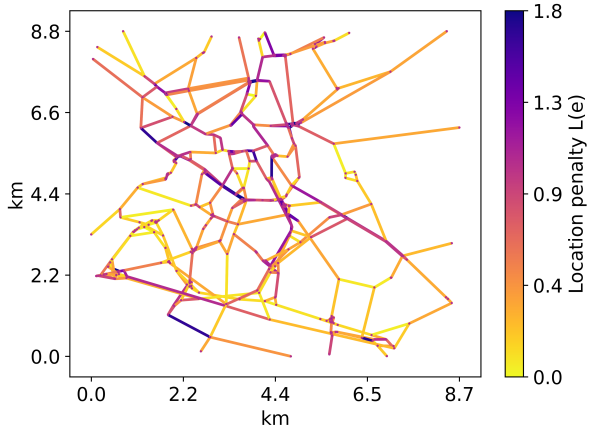
For a given loss model $F(L,T)$ we can then estimate the travel time across an edge e given a departure time t.

$$\begin{aligned} \hat{\tau}(e,t) &= \tau_{min}(e) + \tau_{min}(e) \cdot F(L(e), T(t))) \\ &= \tau_{min}(e) \cdot (1 + F(L(e), T(t))) \end{aligned}$$

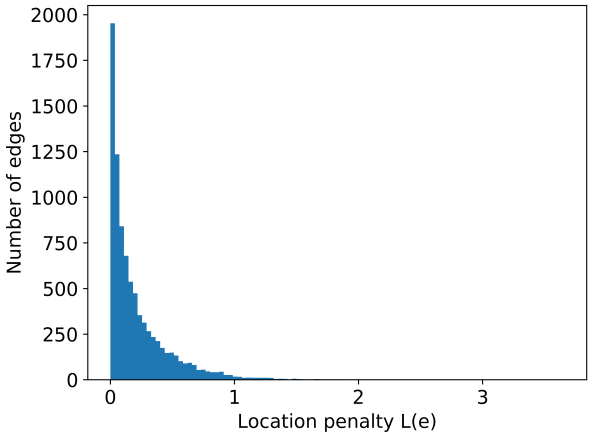
Thus, $\hat{\tau}(e,t)$ is the minimal time plus the estimated delay.

We evaluate 5 different loss models, which are shown in table I. The description of model 0 is $F_0 = 0$. Here the time-dependent part disappears and we get a time-independent routing problem, so F_0 is equivalent to Dijkstra.

The models are constructed from the assumption that time and location penalties reinforce each other. As a first model, we propose their product $L \cdot T$. The second model is similar, but with an intercept term. We expect the intercept coefficient to be zero, but it is worth checking whether our premises hold. Model 3 is a full linear model with 2 parameters. Since we do not yet know if the penalty-error relationship is linear, model 4 and 5 are non-linear. The fourth model is the full quadratic 2-parameter model. Model 5 is an exponential model. To avoid numerical errors, we only allow positive values for the exponents. Furthermore, since Dijkstra requires positive weights, so does TLPM. Therefore, every heuristic returns the maximum of its value and 0.



(a) Location penalties per edge in Ghent.

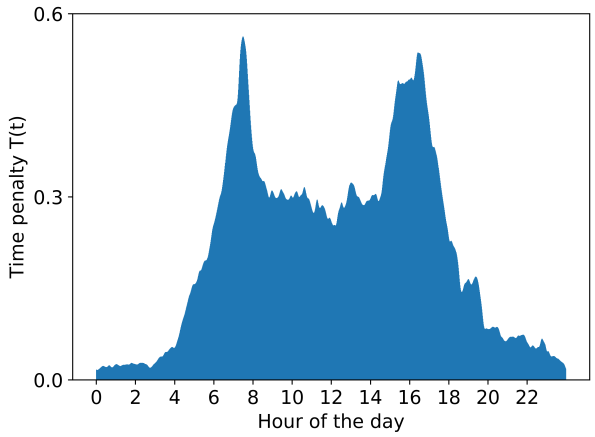


(b) Location penalty distribution in East-Flanders

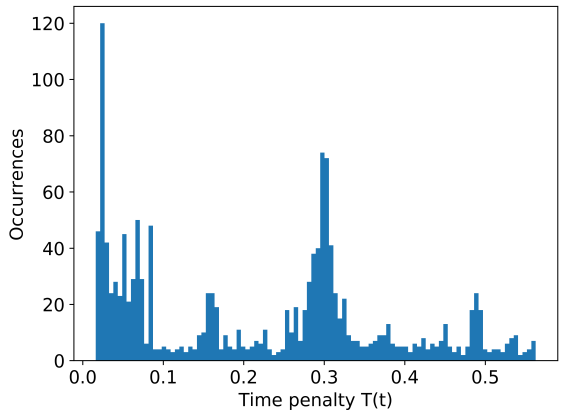
Fig. 2: Location penalties

IV. Experiments

Now we calculate parameters for each of the discussed models. We implemented the algorithms in Python 3.6 (Python Software Foundation, `https://www.python.org/`), using two packages: Matplotlib [8] for the generation of the plots and SciPy for the differential evolution [9]. The algorithm for the differential evolution is due to Storn and Price [10].



(a) Time penalty per minute in East-Flanders



(b) Time penalty distribution in East-Flanders

Fig. 3: Time penalties

A. Single route

For a source node n_s, a target node n_t and a departure time t, the route with minimal travel time, generated by TD-Dijkstra is defined as $R_{min}(n_s, n_t, t)$. Now let F be a time-dependent routing model and $R_F(n_s, n_t, t)$ the route it generates. Given a route R and a departure time t, we can then calculate $\tau(R, t)$: the actual time it takes to travel across R departing at t. Then for a triplet (n_s, n_t, t) we calculate the relative error for a single route:

$$Err(F) := \frac{\tau(R_F, t) - \tau(R_{min}, t)}{\tau(R_{min}, t)}$$

where R_F and R_{min} depend on (n_s, n_t, t). Since TD-Dijkstra returns routes with minimal travel time, Err is always positive.

Such a random route can take many forms. During night or day, in the city of Ghent or its surroundings. Thus we should evaluate several routes to get a more accurate estimate of the quality of the loss model. A single execution of Dijkstra (or TLPM) calculates many routes, all starting from the same node and at the same time, but towards different targets. We can exploit this. Each execution of TLPM, we calculate several hundred routes, instead of one, then we average the errors. This significantly decreases the variance on the outcomes, therefore requiring fewer executions of TLPM to get reliable results.

B. Differential Evolution

Searching the best parameters for each model is a hard task, so we employ a proven method: differential evolution. It is a metaheuristic that optimizes the parameters of a function by generating a population of possible parameter values, calculating the associated function values and then iterating the population until convergence. Thus, this function, referred to as the fitness function, takes as input our model with predefined parameters and must return one value. A single fitness function evaluation consists of running 360 TLPM instances, each instance calculating 360 routes (multiple targets). Then, the return value of our fitness function is the average cost of these $360 \cdot 360$ routes. Each evaluation of this fitness function - which is optimized and parallelized - takes 0.7 seconds. Thanks to the multiple-routes technique and because we use the same random set of routes each evaluation, our fitness function is fairly accurate. However, there is neither gradient-information that could speed up the search nor any guarantee on accuracy. Therefore, we rely on differential evolution, since it is specifically designed as a general-purpose metaheuristic for noisy, real-valued functions.

Differential evolution starts with a population of agents, randomly assigned to the search space. Each agent p is a position vector $[p_1, \ldots, p_{dim}]$ of parameter values. Then for every agent x: first pick 3 random agents a, b, c, different from each other and x. Next initialize a trial-agent y:

$$y_i = \begin{cases} a_i + F \cdot (b_i - c_i) & \text{if } i = R \text{ or } r_i < CR \\ x_i & \text{else} \end{cases}$$

with $CR \in [0, 1]$ and $F \in [0, 2]$ parameters of the differential evolution. Each $r_i \in [0, 1]$ is chosen uniformly random, so the cross-over probability CR decides how often elements of x are replaced by new ones. F, the differential weight or mutation factor, decides how much the replacement differs from a. $R \in [0, 1]$ is randomly chosen and assures that $y \neq x$. If the fitness of the trial-agent y outperforms the fitness of the original agent x, x is replaced by y.

This selection and replacement procedure is repeated for the entire population. This is called an iteration. Finally, iterations are performed until the population converges, or a predefined number of iterations have passed.

The final behavioral parameter is the population size NP. This is the size of the original (and final) population. Furthermore, this is the random/1/binomial variant of differential evolution, because a is a single random vector and the probability distribution of $r_i < CR$ is binomial. Then remains the choice of NP, CR, and F. The optimization of these parameters is extensively described in the thesis of Pedersen [11]. Even more convenient is the table given by Pedersen in [12]. From there we can extract the numbers seen in table II.

TABLE II: Behavioral parameters for DE

Problem Dimensions	Fitness Evaluations	DE Parameters		
		NP	CR	F
2	4.000	24	0.2515	0.8905
5	10.000	20	0.6938	0.9314

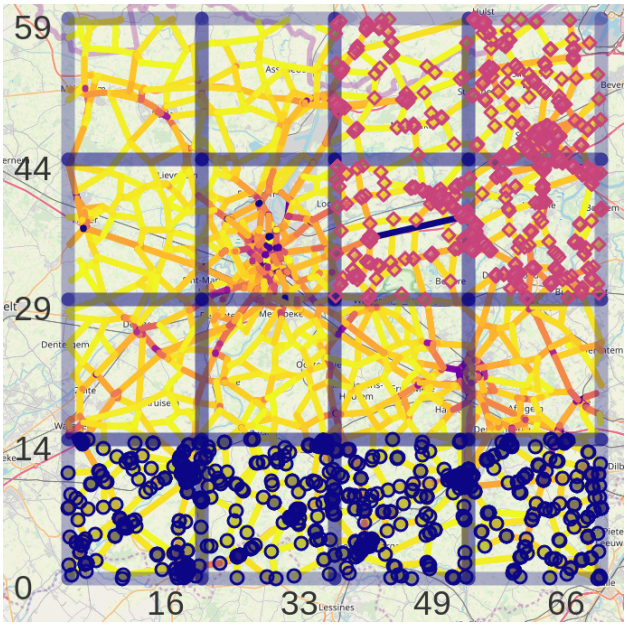


Fig. 4: In blue circles: the bottom rural area. In red diamonds: the second rural area. Ghent is located inside the second square of main diagonal. The axes denote the distance in km.

Once we have good parameters for each model, we will measure the quality of each model with an accuracy test on 10.000 random routes for both the Ghent dataset and the entire East-Flanders network. Then we pick the best model and use this for the final experiment.

C. Areas

We evaluate the best model in multiple regions in East-Flanders. We assess four regions, see table III and fig. 4. Two rather rural regions: the southern part of East-Flanders, bordering with Wallonia (blue circles) and the north-east region around Sint-Niklaas (red diamonds). Both have an area of around $950\,\mathrm{km}^2$. Note that rural here is meant in distinction with city-like, for example, Sint-Niklaas has a population of 80.000. The other two areas that we will evaluate are Ghent, on which we trained the model and the entire time-dependent dataset: a graph of East-Flanders. In table III we notice that there is a large difference between the areas with regards to the average penalty. For instance, in Ghent, it is nearly three times larger than in the southern rural area.

TABLE III: Areas

Area	Mean Penalty	Graph	
		Nodes	Edges
Rural-S	0.13	776	1930
Rural-NE	0.21	802	1796
Ghent	0.37	415	833
East-Flanders	0.22	3825	8570

TABLE IV: The best model parameter values according to the differential evolution search

	Parameter values						Err
Model	a	b	c	d	e	f	(%)
1	2.89						3.68
2	1.75	7.34					3.70
3	4.74	2.10	−4.24	4.36			3.64
4	7.44	0.40	−8.49	8.71	2.69	6.12	3.67
5	2.68	3.56	−0.46	0.86	−0.06	0.30	3.74

V. Results

We calculated parameters for 5 models, by using 360 sources and 360 targets for each parameter set. The number of function evaluations required to reach convergence differed strongly between models. Model 1 only needed 250 function evaluations, while model 5 needed 7.500. We found the following accuracies: see table IV. Often the coefficients of the $L \cdot T$ parameter have the highest value. Moreover, in model 3 and 4, the time penalty T gets a negative modifier so that driving at a rush hour is no problem as long as you drive at a seldom congested road.

The relative error is very similar for each model. With model 3 slightly outperforming the others. Next, we look at the results of the accuracy tests, see fig. 5. These were run for the Ghent dataset (with different routes than during training) and the East-Flanders dataset. We observe that our models outperform the time-independent Dijkstra and that they all have similar results. This time, model 1 has the highest accuracy. Since it is also by far the simplest model, we will continue with it. Thus we find that the best TLPM model is $F = 2.89 \cdot L \cdot T$.

When comparing this model with Dijkstra and TD-Dijkstra, the runtime complexity for each model is $O(|V| \log(|V|))$ if memory-access is constant (which might be problematic for TD-Dijkstra). The memory required for each model is shown in table V. TLPM uses barely more

TABLE V: Comparison between Dijkstra and model 1 in Ghent

Algorithm	Err (%)	Memory-usage
Dijkstra	4.4	$\lvert V\rvert + \lvert E\rvert$
TD-Dijkstra	0	$\lvert V\rvert + \lvert E\rvert \cdot 1440$
TLPM	3.2	$\lvert V\rvert + \lvert E\rvert \cdot 2 + 1440$

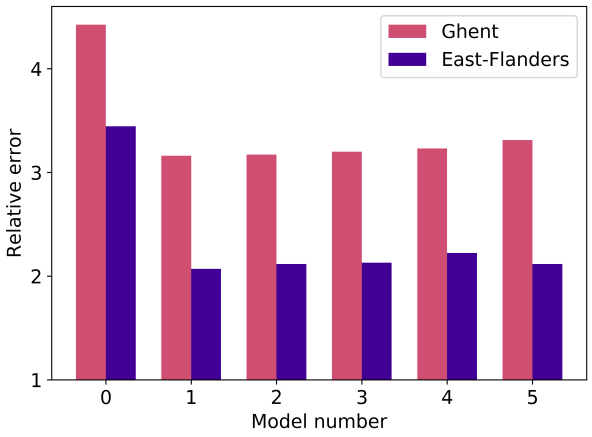


Fig. 5: Results for all models for the relative error tests on the Ghent and East-Flanders datasets for 10.000 routes.

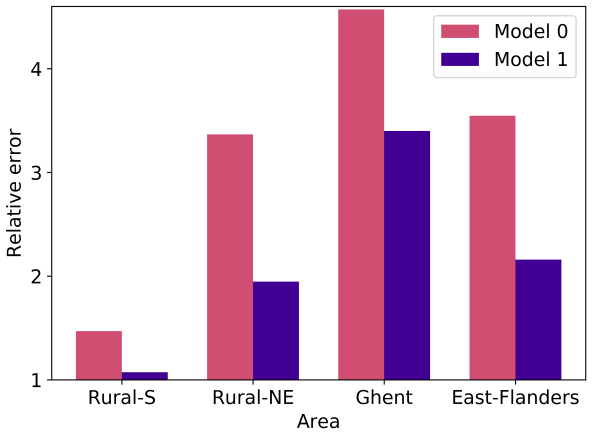


Fig. 6: Dijkstra (model 0) versus model 1 on the different areas for 10.000 routes.

memory than Dijkstra, on the other hand, TD-Dijkstra requires much more. We assumed the availability of a single day of time-data with entries for each minute. The availability of more or less time-data entries directly influences the memory requirements of TD-Dijkstra and the accuracy of both TLPM and TD-Dijkstra. Finally, in every evaluated area, model 1 shows higher accuracy than Dijkstra. Surprisingly, the difference is more pronounced for the rural NE area than for the Ghent area, on which the model was trained. This might be due to the simplicity of the chosen model and the lower variability of delays in the rural NE area. As expected, though, the relative error of both Dijkstra and model 1 decrease when the average penalty decreases.

Conclusion

We implemented a simple memory-efficient time-dependent routing heuristic: TLPM. The model is an adaptation of TD-Dijkstra, with an additional linear-time preprocessing step. TLPM has a better relative error than Dijkstra in all tested settings, although the reduction of the relative error is smaller in more rural areas. Compared to TD-Dijkstra, TLPM requires a small amount of memory. Thus, TLPM elucidates the trade-off between memory and accuracy. Furthermore, it provides an alternate view on time-dependent routing and its intricacies.

We remark that the time penalty depends on the network size. So, when calculating the fastest routes on a new network, the optimization should be repeated. However, in practice, the model is robust and can be extended to most networks without any problems. Furthermore, TLPM is presumed to perform well in dynamic networks. While repeating the full optimization would be inefficient, this is most likely not necessary. Moreover, it is straightforward to update the time and location penalties immediately and only based on the changed values.

Lastly, scalability in time should not be a problem - the bottleneck for both the optimization and the running time is the Dijkstra execution time. The model performance is expected to stay similar to the current results because the dataset used in testing was already diverse (a variety of rural and city-like environments). Furthermore, an extensive part of the model is based on local values, which contain the same information in larger graphs. In light of these good expectations, effectively testing the model on larger and different types of graphs is an important next step.

Acknowledgment

This research was partly funded by the Ghent University IOP project “Modelling Uncertainty in Hub Location Planning through Interdisciplinary Research”.

The 2014 dataset was provided by Be-Mobile for research purposes. At the time of publication, they have already significantly advanced their datasets.

References

[1] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck, “Route Planning in Transportation Networks,” *arXiv:1504.05140 [cs]*, Apr. 2015.

[2] G. V. Batz, D. Delling, P. Sanders, and C. Vetter, “Time-Dependent Contraction Hierarchies,” *2009 Proceedings of the Eleventh Workshop on Algorithm Engineering and Experiments (ALENEX)*, pp. 97–105, Jan. 2009.

[3] G. V. E. Batz, “Time-Dependent Route Planning with Contraction Hierarchies,” Ph.D. dissertation, Karlsruhe Institute of Technology, Karlsruhe, 2014.

[4] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck, “Customizable Route Planning in Road Networks,” *Transportation Science*, vol. 51, no. 2, pp. 566–591, May 2015.

[5] B. Strasser, “Intriguingly Simple and Efficient Time-Dependent Routing in Road Networks,” *arXiv:1606.06636 [cs]*, June 2016.

[6] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[7] S. E. Dreyfus, “An Appraisal of Some Shortest-Path Algorithms,” *Operations Research*, vol. 17, no. 3, pp. 395–412, June 1969.

[8] J. D. Hunter, “Matplotlib: A 2D Graphics Environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[9] E. Jones, T. Oliphant, P. Peterson, and others, “SciPy: Open source scientific tools for Python,” 2001.

[10] R. Storn and K. Price, “Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces,” *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, Nov. 1996.

[11] M. E. H. Pedersen, “Tuning & Simplifying Heuristical Optimization,” Ph.D. dissertation, University of Southampton, Jan. 2010.

[12] ——, “Good Parameters for Differential Evolution,” Hvass Laboratories, Technical Report HL1002, 2010.